SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

ZIRCUIT LABS

Overview

Project Summary

- Name: Zircuit Labs - Zkr Staking
- Platform: Zircuit
- Language: Solidity
- Repository:
    - https://github.com/zircuit-labs/zkr-staking
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Zircuit Labs - Zkr Staking
Version	v3
Type	Solidity
Dates	Sep 14 2024
Logs	Aug 29 2024; Sep 02 2024; Sep 14 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	1
Total	2

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Inconsistencies in records caused by non-single entry points	Low	Business Logic	Acknowledged
2	Custom error don’t contain revert reason	Informational	Code Quality	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Inconsistencies in records caused by non-single entry points

Severity: Low	Category: Business Logic
Target: - contracts/LiquidityHub.sol - contracts/BasePool.sol	

Description

There are five functions to deposit tokens to Launpool. The Readme file mentions that this is done by keeping track of the total deposits through those functions in the `totalBalances` mapping. So the `totalBalances` is equal to the corresponding pool's total deposit amount.

contracts/LiquidityHub.sol:L49, L58, L91, L146, L185

```
function depositToLaunchPool(...)
function depositToLaunchPoolWithSig(...)
function withdrawMerkle(...)
function depositToLaunchPoolMerkle(...)
function depositToLaunchPoolWithSigMerkle(...)
```

The `Launpool` is inherited from `BasePool`, so it also has the `depositFor()` and the `withdraw()` functions. If the user deposits to `Launpool` directly. The pool's total deposit amount will increase but `LiquidityHub` will not. This kind of equality relationship will be broken. And the record inconsistency will occur.

contracts/BasePool.sol:L61, L116

```
function depositFor(...)
function withdraw(...)
```

Recommendation

By prohibiting direct access to deposit/withdraw the pool, a unified entry point is established.

Status

The issue has been acknowledged by the team. The team has stated that `LiquidityHub.totalBalance` == `Liquidity Hub total deposit` - (`LiquidityHub Withdrawals` + `Liquidity Transfers to LaunchPool`).

2.3 Informational Findings

2. Custom error don’t contain revert reason

Severity: Informational	Category: Code Quality
Target: - contracts/LaunchPool.sol	

Description

When a call to contractAddress has failed, the user could not understand the reason for the revert, because custom error `CallExecutionFailed()` doesn't have a bytes result.

contracts/LaunchPool.sol:L162-L188

```
function _executeCall(
    address _user,
    ContractCall calldata _contractCall
) internal returns (bytes memory){
    ...

    (bool success, bytes memory result) = _contractCall.contractAddress.call(
        abi.encodePacked(_contractCall.functionSignature, abi.encode(_user,
_contractCall.data))
    );

    if(!success) revert CallExecutionFailed();
}
```

Recommendation

Consider Returning the call result.

```
if(!success) revert CallExecutionFailed(result);
```

Status

The issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 58a0406:

File	SHA-1 hash
contracts/LiquidityHub.sol	a1e4b17a6d1e8b72a5d8522b7a55e06f10938c8f
contracts/LaunchPool.sol	585b1c15998875ff68781572292e4f9cf81b60c3
contracts/BasePool.sol	5a14e5848558d8c3de1a2382dab8cd1c8bdad2ac
contracts/LaunchPoolFactory.sol	ca18de484c5255318f78bd122a355ecc752b8905

And we audited the commit dad0eff that introduced new features zircuit-labs/zkr-staking repository:

File	SHA-1 hash
contracts/BasePool.sol	d3df6a886f0925d50d0828d5591d8399ed6481cf
contracts/LaunchPool.sol	efdb45cf4b852842c964b0bc43bc68af069885b7